Índice

1. algorithm

#include <algorithm> #include <numeric>

Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i ∈[f,l) tq. i=elem, pred(i), i∈[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) ∈ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, ...)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i$ ∈[f,l)
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2. Estructuras

2.1. RMQ (static)

Dado un arreglo y una operacion asociativa *idempotente*, get(i, j) opera sobre el rango [i, j). Restriccion: LVL ≥ ceil(logn); Usar [ ] para llenar arreglo y luego build().

```
1  struct RMQ{
2    #define LVL 10
3    tipo vec[LVL][1<<(LVL+1)];
4    tipo &operator[](int p){return vec[0][p];}
5    tipo get(int i, int j) {//intervalo [i,j)
6      int p = 31-__builtin_clz(j-i);
7      return min(vec[p][i],vec[p][j-(1<<p)]);
8    }
9    void build(int n) {//O(nlogn)
10     int mp = 31-__builtin_clz(n);
11     forn(p, mp) forn(x, n-(1<<p))
12       vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
13   }};
```

2.2. RMQ (dynamic)

```
1  //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
       sobre el rango [i, j).
2  #define MAXN 100000
3  #define operacion(x, y) max(x, y)
4  const int neutro=0;
5  struct RMQ{
6    int sz;
7    tipo t[4*MAXN];
8    tipo &operator[](int p){return t[sz+p];}
9    void init(int n){//O(nlgn)
10     sz = 1 << (32-__builtin_clz(n));
11     forn(i, 2*sz) t[i]=neutro;
12   }
13   void updall(){//O(n)
14     dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);}
15   tipo get(int i, int j){return get(i,j,1,0,sz);}
16   tipo get(int i, int j, int n, int a, int b){//O(lgn)
17     if(j<=a || i>=b) return neutro;
18     if(i<=a && b<=j) return t[n];
19     int c=(a+b)/2;
```

```
20     return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
21   }
22   void set(int p, tipo val){//O(lgn)
23     for(p+=sz; p>0 && t[p]!=val;){
24       t[p]=val;
25       p/=2;
26       val=operacion(t[p*2], t[p*2+1]);
27     }
28   }
29 }rmq;
30 //Usage:
31 cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

2.3. RMQ (lazy)

```
1  //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera
       sobre el rango [i, j).
2  typedef int Elem;//Elem de los elementos del arreglo
3  typedef int Alt;//Elem de la alteracion
4  #define operacion(x,y) x+y
5  const Elem neutro=0; const Alt neutro2=0;
6  #define MAXN 100000
7  struct RMQ{
8    int sz;
9    Elem t[4*MAXN];
10   Alt dirty[4*MAXN];//las alteraciones pueden ser de distinto Elem
11   Elem &operator[](int p){return t[sz+p];}
12   void init(int n){//O(nlgn)
13     sz = 1 << (32-__builtin_clz(n));
14     forn(i, 2*sz) t[i]=neutro;
15     forn(i, 2*sz) dirty[i]=neutro2;
16   }
17   void push(int n, int a, int b){//propaga el dirty a sus hijos
18     if(dirty[n]!=0){
19       t[n]+=dirty[n]*(b-a);//altera el nodo
20       if(n<sz){
21         dirty[2*n]+=dirty[n];
22         dirty[2*n+1]+=dirty[n];
23       }
24       dirty[n]=0;
25     }
26   }
27   Elem get(int i, int j, int n, int a, int b){//O(lgn)
```

```
28      if(j<=a || i>=b) return neutro;
29      push(n, a, b);//corrige el valor antes de usarlo
30      if(i<=a && b<=j) return t[n];
31      int c=(a+b)/2;
32      return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
33    }
34    Elem get(int i, int j){return get(i,j,1,0,sz);}
35    //altera los valores en [i, j) con una alteracion de val
36    void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)
37      push(n, a, b);
38      if(j<=a || i>=b) return;
39      if(i<=a && b<=j){
40        dirty[n]+=val;
41        push(n, a, b);
42        return;
43      }
44      int c=(a+b)/2;
45      alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
46      t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de arriba
47    }
48    void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
49  }rmq;
```

2.4.   RMQ (persistente)

```
1  typedef int tipo;
2  tipo oper(const tipo &a, const tipo &b){
3      return a+b;
4  }
5  struct node{
6    tipo v; node *l,*r;
7    node(tipo v):v(v), l(NULL), r(NULL) {}
8      node(node *l, node *r) : l(l), r(r){
9          if(!l) v=r->v;
10         else if(!r) v=l->v;
11         else v=oper(l->v, r->v);
12      }
13 };
14 node *build (tipo *a, int tl, int tr) {//modificar para que tome tipo a
15   if (tl+1==tr) return new node(a[tl]);
16   int tm=(tl + tr)>>1;
17   return new node(build(a, tl, tm), build(a, tm, tr));
18 }
```

```
19 node *update(int pos, int new_val, node *t, int tl, int tr){
20   if (tl+1==tr) return new node(new_val);
21   int tm=(tl+tr)>>1;
22   if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r)
          ;
23   else return new node(t->l, update(pos, new_val, t->r, tm, tr));
24 }
25 tipo get(int l, int r, node *t, int tl, int tr){
26     if(l==tl && tr==r) return t->v;
27   int tm=(tl + tr)>>1;
28     if(r<=tm) return get(l, r, t->l, tl, tm);
29     else if(l>=tm) return get(l, r, t->r, tm, tr);
30   return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
31 }
```

2.5.   Fenwick Tree

```
1  //For 2D threat each column as a Fenwick tree, by adding a nested for in
        each operation
2  struct Fenwick{
3    static const int sz=1000001;
4    tipo t[sz];
5    void adjust(int p, tipo v){//valid with p in [1, sz), O(lgn)
6      for(; p<sz; p+=(p&-p)) t[p]+=v; }
7    tipo sum(int p){//cumulative sum in [1, p], O(lgn)
8      tipo s=0;
9      for(; p; p-=(p&-p)) s+=t[p];
10     return s;
11   }
12   tipo sum(int a, int b){return sum(b)-sum(a-1);}
13   //get largest value with cumulative sum less than or equal to x;
14   //for smallest, pass x-1 and add 1 to result
15   int getind(tipo x) {//O(lgn)
16       int idx = 0, mask = N;
17       while(mask && idx < N) {
18         int t = idx + mask;
19       if(x >= tree[t])
20           idx = t, x -= tree[t];
21       mask >>= 1;
22       }
23       return idx;
24   }};
```

2.6.   Union Find

```
struct UnionFind{
  vector<int> f;//the array contains the parent of each node
  void init(int n){f.clear(); f.insert(f.begin(), n, -1);}
  int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));}//O(1)
  bool join(int i, int j) {
    bool con=comp(i)==comp(j);
    if(!con) f[comp(i)] = comp(j);
    return con;
  }};
```

2.7.   Disjoint Intervals

```
bool operator< (const ii &a, const ii &b) {return a.fst<b.fst;}
//Stores intervals as [first, second]
//in case of a collision it joins them in a single interval
struct disjoint_intervals {
  set<ii> segs;
  void insert(ii v) {//O(lgn)
    if(v.snd-v.fst==0.) return;//OJO
    set<ii>::iterator it,at;
    at = it = segs.lower_bound(v);
    if (at!=segs.begin() && (--at)->snd >= v.fst)
      v.fst = at->fst, --it;
    for(; it!=segs.end() && it->fst <= v.snd; segs.erase(it++))
      v.snd=max(v.snd, it->snd);
    segs.insert(v);
  }
};
```

2.8.   RMQ (2D)

```
struct RMQ2D{
  static const int sz=1024;
  RMQ t[sz];
  RMQ &operator[](int p){return t[sz/2+p];}
  void build(int n, int m){//O(nm)
    forr(y, sz/2, sz/2+m)
      t[y].build(m);
    forr(y, sz/2+m, sz)
      forn(x, sz)
        t[y].t[x]=0;
    dforn(y, sz/2)
```

```
      forn(x, sz)
        t[y].t[x]=max(t[y*2].t[x], t[y*2+1].t[x]);
  }
  void set(int x, int y, tipo v){//O(lgm.lgn)
    y+=sz/2;
    t[y].set(x, v);
    while(y/=2)
      t[y].set(x, max(t[y*2][x], t[y*2+1][x]));
  }
  //O(lgm.lgn)
  int get(int x1, int y1, int x2, int y2, int n=1, int a=0, int b=sz/2){
    if(y2<=a || y1>=b) return 0;
    if(y1<=a && b<=y2) return t[n].get(x1, x2);
    int c=(a+b)/2;
    return max(get(x1, y1, x2, y2, 2*n, a, c),
        get(x1, y1, x2, y2, 2*n+1, c, b));
  }
};
//Example to initialize a grid of M rows and N columns:
RMQ2D rmq;
forn(i, M)
  forn(j, N)
    cin >> rmq[i][j];
rmq.build(N, M);
```

2.9.   Big Int

```
#define BASEXP 6
#define BASE 1000000
#define LMAX 1000
struct bint{
    int l;
    ll n[LMAX];
    bint(ll x=0){
        l=1;
        forn(i, LMAX){
            if (x) l=i+1;
            n[i]=x%BASE;
            x/=BASE;

        }
    }
    bint(string x){
```

```
17        l=(x.size()-1)/BASEXP+1;
18            fill(n, n+LMAX, 0);
19            ll r=1;
20            forn(i, sz(x)){
21                n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
22                r*=10; if(r==BASE)r=1;
23            }
24        }
25        void out(){
26        cout << n[l-1];
27        dforn(i, l-1) printf("%.6llu", n[i]);//6=BASEXP!
28        }
29      void invar(){
30        fill(n+l, n+LMAX, 0);
31        while(l>1 && !n[l-1]) l--;
32      }
33  };
34  bint operator+(const bint&a, const bint&b){
35    bint c;
36      c.l = max(a.l, b.l);
37      ll q = 0;
38      forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q %BASE, q/=BASE;
39      if(q) c.n[c.l++] = q;
40      c.invar();
41      return c;
42  }
43  pair<bint, bool> lresta(const bint& a, const bint& b)    // c = a - b
44  {
45    bint c;
46      c.l = max(a.l, b.l);
47      ll q = 0;
48      forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE) %BASE, q=(q+BASE)/
              BASE-1;
49      c.invar();
50      return make_pair(c, !q);
51  }
52  bint& operator-= (bint& a, const bint& b){return a=lresta(a, b).first;}
53  bint operator- (const bint&a, const bint&b){return lresta(a, b).first;}
54  bool operator< (const bint&a, const bint&b){return !lresta(a, b).second
        ;}
55  bool operator<= (const bint&a, const bint&b){return lresta(b, a).second
        ;}
56  bool operator==(const bint&a, const bint&b){return a <= b && b <= a;}
57  bint operator*(const bint&a, ll b){
58      bint c;
59      ll q = 0;
60      forn(i, a.l) q += a.n[i]*b, c.n[i] = q %BASE, q/=BASE;
61      c.l = a.l;
62      while(q) c.n[c.l++] = q %BASE, q/=BASE;
63      c.invar();
64      return c;
65  }
66  bint operator*(const bint&a, const bint&b){
67      bint c;
68      c.l = a.l+b.l;
69      fill(c.n, c.n+b.l, 0);
70      forn(i, a.l){
71          ll q = 0;
72          forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q %BASE, q
              /=BASE;
73          c.n[i+b.l] = q;
74      }
75      c.invar();
76      return c;
77  }
78  pair<bint, ll> ldiv(const bint& a, ll b){// c = a / b ; rm = a % b
79    bint c;
80    ll rm = 0;
81    dforn(i, a.l){
82          rm = rm * BASE + a.n[i];
83          c.n[i] = rm / b;
84          rm %= b;
85      }
86      c.l = a.l;
87      c.invar();
88      return make_pair(c, rm);
89  }
90  bint operator/(const bint&a, ll b){return ldiv(a, b).first;}
91  ll operator%(const bint&a, ll b){return ldiv(a, b).second;}
92  pair<bint, bint> ldiv(const bint& a, const bint& b){
93    bint c;
94      bint rm = 0;
95      dforn(i, a.l){
96          if (rm.l==1 && !rm.n[0])
97              rm.n[0] = a.n[i];
98          else{
```

```
99              dforn(j, rm.l) rm.n[j+1] = rm.n[j];
100             rm.n[0] = a.n[i];
101             rm.l++;
102         }
103         ll q = rm.n[b.l] * BASE + rm.n[b.l-1];
104         ll u = q / (b.n[b.l-1] + 1);
105         ll v = q /  b.n[b.l-1] + 1;
106         while (u < v-1){
107             ll m = (u+v)/2;
108             if (b*m <= rm) u = m;
109             else v = m;
110         }
111         c.n[i]=u;
112         rm-=b*u;
113     }
114   c.l=a.l;
115     c.invar();
116     return make_pair(c, rm);
117 }
118 bint operator/(const bint&a, const bint&b){return ldiv(a, b).first;}
119 bint operator%(const bint&a, const bint&b){return ldiv(a, b).second;}
```

2.10. HashTables

```
1  //Compilar: g++ --std=c++11
2  struct Hash{
3    size_t operator()(const ii &a)const{
4      size_t s=hash<int>()(a.fst);
5      return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
6    }
7    size_t operator()(const vector<int> &v)const{
8      size_t s=0;
9      for(auto &e : v)
10        s ^= hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
11     return s;
12   }
13 };
14 unordered_set<ii, Hash> s;
15 unordered_map<ii, int, Hash> m;//map<key, value, hasher>
```

2.11. Modnum

```
1  struct mnum{
2    static const tipo mod=12582917;
```

```
3    tipo v;
4    mnum(tipo v=0): v(v%mod) {}
5    mnum operator+(mnum b){return v+b.v;}
6    mnum operator-(mnum b){return v>=b.v? v-b.v : mod-b.v+v;}
7    mnum operator*(mnum b){return v*b.v;}
8    mnum operator^(int n){
9      if(!n) return 1;
10     return n%2? (*this)^(n/2)*(this) : (*this)^(n/2);}
11 };
```

2.12. Treap

```
1  typedef int Key;
2  typedef struct node *pnode;
3  struct node{
4      Key key;
5      int prior, size;
6      pnode l,r;
7      node(Key key=0, int prior=0): key(key), prior(prior), size(1), l(0),
8          r(0) {}
8  };
9  struct treap {
10     pnode root;
11     treap(): root(0) {}
12     int size(pnode p) { return p ? p->size : 0; }
13     int size() { return size(root); }
14     void push(pnode p) {
15         // modificar y propagar el dirty a los hijos aca(para lazy)
16     }
17     // Update function and size from children's values
18     void pull(pnode p) {//recalcular valor del nodo aca (para rmq)
19         p->size = 1 + size(p->l) + size(p->r);
20     }
21     pnode merge(pnode l, pnode r) {
22         if (!l || !r) return l ? l : r;
23         push(l), push(r);
24         pnode t;
25         if (l->prior < r->prior) l->r=merge(l->r, r), t = l;
26         else r->l=merge(l, r->l), t = r;
27         pull(t);
28         return t;
29     }//opcional:
30     void merge(treap t) {root = merge(root, t.root), t.root=0;}
```

```
31    //*****KEY OPERATIONS*****//
32    void splitKey(pnode t, Key key, pnode &l, pnode &r) {
33        if (!t) return void(l = r = 0);
34        push(t);
35        if (key <= t->key) splitKey(t->l, key, l, t->l), r = t;
36        else splitKey(t->r, key, t->r, r), l = t;
37        pull(t);
38    }
39    void insertKey(Key key) {
40        pnode elem = new node(key, rand());
41        pnode t1, t2; splitKey(root, key, t1, t2);
42        t1=merge(t1,elem);
43        root=merge(t1,t2);
44    }
45    void eraseKeys(Key key1, Key key2) {
46        pnode t1,t2,t3;
47        splitKey(root,key1,t1,t2);
48        splitKey(t2,key2, t2, t3);
49        root=merge(t1,t3);
50    }
51    void eraseKey(pnode &t, Key key) {
52        if (!t) return;
53        push(t);
54        if (key == t->key) t=merge(t->l, t->r);
55        else if (key < t->key) eraseKey(t->l, key);
56        else eraseKey(t->r, key);
57        pull(t);
58    }
59    void eraseKey(Key key) {eraseKey(root, key);}
60    pnode findKey(pnode t, Key key) {
61        if (!t) return 0;
62        if (key == t->key) return t;
63        if (key < t->key) return findKey(t->l, key);
64        return findKey(t->r, key);
65    }
66    pnode findKey(Key key) { return findKey(root, key); }
67    //*****POS OPERATIONS*****// No mezclar con las funciones Key
68    //(No funciona con pos:)
69    void splitSize(pnode t, int sz, pnode &l, pnode &r) {
70        if (!t) return void(l = r = 0);
71        push(t);
72        if (sz <= size(t->l)) splitSize(t->l, sz, l, t->l), r = t;
73        else splitSize(t->r, sz - 1 - size(t->l), t->r, r), l = t;
```

```
74        pull(t);
75    }
76    void insertPos(int pos, Key key) {
77        pnode elem = new node(key, rand());
78        pnode t1,t2; splitSize(root, pos, t1, t2);
79        t1=merge(t1,elem);
80        root=merge(t1,t2);
81    }
82    void erasePos(int pos1, int pos2=-1) {
83    if(pos2==-1) pos2=pos1+1;
84        pnode t1,t2,t3;
85        splitSize(root,pos1,t1,t2);
86        splitSize(t2,pos2-pos1,t2,t3);
87        root=merge(t1, t2);
88    }
89    pnode findPos(pnode t, int pos) {
90        if(!t) return 0;
91        if(pos <= size(t->l)) return findPos(t->l, pos);
92        return findPos(t->r, pos - 1 - size(t->l));
93    }
94    Key &operator[](int pos){return findPos(root, pos)->key;}//ojito
95 };
```

2.13. Convex Hull Trick

```
1  const ll is_query = -(1LL<<62);
2  struct Line {
3      ll m, b;
4      mutable multiset<Line>::iterator it;
5      const Line *succ(multiset<Line>::iterator it) const;
6      bool operator<(const Line& rhs) const {
7          if (rhs.b != is_query) return m < rhs.m;
8          const Line *s=succ(it);
9          if(!s) return 0;
10         ll x = rhs.m;
11         return b - s->b < (s->m - m) * x;
12     }
13 };
14 struct HullDynamic : public multiset<Line>{ // will maintain upper hull
       for maximum
15     bool bad(iterator y) {
```

```
16        iterator z = next(y);
17        if (y == begin()) {
18            if (z == end()) return 0;
19            return y->m == z->m && y->b <= z->b;
20        }
21        iterator x = prev(y);
22        if (z == end()) return y->m == x->m && y->b <= x->b;
23        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m
              );
24    }
25    iterator next(iterator y){return ++y;}
26    iterator prev(iterator y){return --y;}
27    void insert_line(ll m, ll b) {
28        iterator y = insert((Line) { m, b });
29        y->it=y;
30        if (bad(y)) { erase(y); return; }
31        while (next(y) != end() && bad(next(y))) erase(next(y));
32        while (y != begin() && bad(prev(y))) erase(prev(y));
33    }
34    ll eval(ll x) {
35        Line l = *lower_bound((Line) { x, is_query });
36        return l.m * x + l.b;
37    }
38 }h;
39 const Line *Line::succ(multiset<Line>::iterator it) const{
40    return (++it==h.end()? NULL : &*it);}
```

2.14. Gain-Cost Set

```
1  //esta estructura mantiene pairs(beneficio, costo)
2  //de tal manera que en el set quedan ordenados
3  //por beneficio Y COSTO creciente. (va borrando los que no son optimos)
4  struct V{
5    int gain, cost;
6    bool operator<(const V &b)const{return gain<b.gain;}
7  };
8  set<V> s;
9  void add(V x){
10   set<V>::iterator p=s.lower_bound(x);//primer elemento mayor o igual
11   if(p!=s.end() && p->cost <= x.cost) return;//ya hay uno mejor
12   p=s.upper_bound(x);//primer elemento mayor
13   if(p!=s.begin()){//borro todos los peores (<=beneficio  y >=costo)
```

```
14     --p;//ahora es ultimo elemento menor o igual
15     while(p->cost >= x.cost){
16       if(p==s.begin()){s.erase(p); break;}
17       s.erase(p--);
18     }
19   }
20   s.insert(x);
21 }
22 int get(int gain){//minimo costo de obtener tal ganancia
23   set<V>::iterator p=s.lower_bound((V){gain, 0});
24   return p==s.end()? INF : p->cost;}
```

2.15. Set con busq binaria

```
1  #include <ext/pb_ds/assoc_container.hpp>
2  #include <ext/pb_ds/tree_policy.hpp>
3  using namespace  __gnu_pbds;
4  typedef tree<int,null_type,less<int>,//key,mapped type, comparator
5      rb_tree_tag,tree_order_statistics_node_update> set_t;
6  //find_by_order(i) devuelve iterador al i-esimo elemento
7  //order_of_key(k): devuelve la pos del lower bound de k
8  //Ej: 12, 100, 505, 1000, 10000.
9  //order_of_key(10) == 0, order_of_key(100) == 1,
10 //order_of_key(707) == 3, order_of_key(9999999) == 5
```

3. Algos

3.1. Longest Increasing Subsecuence

```
1  //Para non-increasing, cambiar comparaciones y revisar busq binaria
2  //Given an array, paint it in the least number of colors so that each
       color turns to a non-increasing subsecuence.
3  //Solution:Min number of colors=Length of the longest increasing
       subsecuence
4  int N, a[MAXN];//secuencia y su longitud
5  ii d[MAXN+1];//d[i]=ultimo valor de la subsecuencia de tamanio i
6  int p[MAXN];//padres
7  vector<int> R;//respuesta
8  void rec(int i){
9    if(i==-1) return;
10   R.push_back(a[i]);
11   rec(p[i]);
12 }
```

```
13  int lis(){//O(nlogn)
14    d[0] = ii(-INF, -1); forn(i, N) d[i+1]=ii(INF, -1);
15    forn(i, N){
16      int j = upper_bound(d, d+N+1, ii(a[i], INF))-d;
17      if (d[j-1].first < a[i]&&a[i] < d[j].first){
18        p[i]=d[j-1].second;
19        d[j] = ii(a[i], i);
20      }
21    }
22    R.clear();
23    dforn(i, N+1) if(d[i].first!=INF){
24      rec(d[i].second);//reconstruir
25      reverse(R.begin(), R.end());
26      return i;//longitud
27    }
28    return 0;
29  }
```

3.2. Manacher

```
1   int d1[MAXN];//d1[i]=long del maximo palindromo impar con centro en i
2   int d2[MAXN];//d2[i]=analogo pero para longitud par
3   //0 1 2 3 4
4   //a a b c c <--d1[2]=3
5   //a a b b <--d2[2]=2 (estan uno antes)
6   void manacher(){
7     int l=0, r=-1, n=sz(s);
8     forn(i, n){
9       int k=(i>r? 1 : min(d1[l+r-i], r-i));
10      while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) ++k;
11      d1[i] = k--;
12      if(i+k > r) l=i-k, r=i+k;
13    }
14    l=0, r=-1;
15    forn(i, n){
16      int k=(i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
17      while(i+k-1<n && i-k>=0 && s[i+k-1]==s[i-k]) k++;
18      d2[i] = --k;
19      if(i+k-1 > r) l=i-k, r=i+k-1;
20    }
```

3.3. Alpha-Beta prunning

```
1   ll alphabeta(State &s, bool player = true, int depth = 1e9, ll alpha = -
        INF, ll beta = INF) { //player = true -> Maximiza
2     if(s.isFinal()) return s.score;
3   //~ if (!depth) return s.heuristic();
4     vector<State> children;
5     s.expand(player, children);
6     int n = children.size();
7     forn(i, n) {
8       ll v = alphabeta(children[i], !player, depth-1, alpha, beta);
9       if(!player) alpha = max(alpha, v);
10      else beta = min(beta, v);
11      if(beta <= alpha) break;
12    }
13    return !player ? alpha : beta;}
```

4.   Strings

4.1.   KMP

```
1   string T;//cadena donde buscar(where)
2   string P;//cadena a buscar(what)
3   int b[MAXLEN];//back table
4   void kmppre(){//by gabina with love
5     int i =0, j=-1; b[0]=-1;
6     while(i<sz(P)){
7       while(j>=0 && P[i] != P[j]) j=b[j];
8       i++, j++;
9       b[i] = j;
10    }
11  }
12
13  void kmp(){
14    int i=0, j=0;
15    while(i<sz(T)){
16      while(j>=0 && T[i]!=P[j]) j=b[j];
17      i++, j++;
18      if(j==sz(P)){
19        printf("P␣is␣found␣at␣index␣%d␣in␣T\n", i-j);
20        j=b[j];
21      }
22    }
23  }
```

4.2.   Trie

```
1  struct trie{
2    map<char, trie> m;
3    void add(const string &s, int p=0){
4      if(s[p]) m[s[p]].add(s, p+1);
5    }
6    void dfs(){
7      //Do stuff
8      forall(it, m)
9        it->second.dfs();
10   }
11 };
```

4.3.   Suffix Array (largo, nlogn)

```
1  #define MAX_N 1000
2  #define rBOUND(x) (x<n? r[x] : 0)
3  //sa will hold the suffixes in order.
4  int sa[MAX_N], r[MAX_N], n;
5  string s; //input string, n=sz(s)
6
7  int f[MAX_N], tmpsa[MAX_N];
8  void countingSort(int k){
9    zero(f);
10   forn(i, n) f[rBOUND(i+k)]++;
11   int sum=0;
12   forn(i, max(255, n)){
13     int t=f[i]; f[i]=sum; sum+=t;}
14   forn(i, n)
15     tmpsa[f[rBOUND(sa[i]+k)]++]=sa[i];
16   memcpy(sa, tmpsa, sizeof(sa));
17 }
18 void constructsa(){//O(n log n)
19   n=sz(s);
20   forn(i, n) sa[i]=i, r[i]=s[i];
21   for(int k=1; k<n; k<<=1){
22     countingSort(k), countingSort(0);
23     int rank, tmpr[MAX_N];
24     tmpr[sa[0]]=rank=0;
25     forr(i, 1, n)
26       tmpr[sa[i]]=(r[sa[i]]==r[sa[i-1]] && r[sa[i]+k]==r[sa[i-1]+k] )?
                rank : ++rank;
```

```
27     memcpy(r, tmpr, sizeof(r));
28     if(r[sa[n-1]]==n-1) break;
29   }
30 }
31 void print(){//for debug
32   forn(i, n)
33     cout << i << '␣' <<
34     s.substr(sa[i], s.find( '$', sa[i])-sa[i]) << endl;}
```

4.4.   String Matching With Suffix Array

```
1  //returns (lowerbound, upperbound) of the search
2  ii stringMatching(string P){ //O(sz(P)lgn)
3    int lo=0, hi=n-1, mid=lo;
4    while(lo<hi){
5      mid=(lo+hi)/2;
6      int res=s.compare(sa[mid], sz(P), P);
7      if(res>=0) hi=mid;
8      else lo=mid+1;
9    }
10   if(s.compare(sa[lo], sz(P), P)!=0) return ii(-1, -1);
11   ii ans; ans.fst=lo;
12   lo=0, hi=n-1, mid;
13   while(lo<hi){
14     mid=(lo+hi)/2;
15     int res=s.compare(sa[mid], sz(P), P);
16     if(res>0) hi=mid;
17     else lo=mid+1;
18   }
19   if(s.compare(sa[hi], sz(P), P)!=0) hi--;
20   ans.snd=hi;
21   return ans;
22 }
```

4.5.   LCP (Longest Common Prefix)

```
1  //Calculates the LCP between consecutives suffixes in the Suffix Array.
2  //LCP[i] is the length of the LCP between sa[i] and sa[i-1]
3  int LCP[MAX_N], phi[MAX_N], PLCP[MAX_N];
4  void computeLCP(){//O(n)
5    phi[sa[0]]=-1;
6    forr(i, 1, n) phi[sa[i]]=sa[i-1];
7    int L=0;
8    forn(i, n){
```

```
9      if(phi[i]==-1) {PLCP[i]=0; continue;}
10     while(s[i+L]==s[phi[i]+L]) L++;
11     PLCP[i]=L;
12     L=max(L-1, 0);
13   }
14   forn(i, n) LCP[i]=PLCP[sa[i]];
15 }
```

4.6.   Corasick

```
1
2 struct trie{
3   map<char, trie> next;
4   trie* tran[256];//transiciones del automata
5   int idhoja, szhoja;//id de la hoja o 0 si no lo es
6   //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que
         es hoja
7   trie *padre, *link, *nxthoja;
8   char pch;//caracter que conecta con padre
9   trie(): tran(), idhoja(), padre(), link() {}
10  void insert(const string &s, int id=1, int p=0){//id>0!!!
11    if(p<sz(s)){
12      trie &ch=next[s[p]];
13      tran[(int)s[p]]=&ch;
14      ch.padre=this, ch.pch=s[p];
15      ch.insert(s, id, p+1);
16    }
17    else idhoja=id, szhoja=sz(s);
18  }
19  trie* get_link() {
20    if(!link){
21      if(!padre) link=this;//es la raiz
22      else if(!padre->padre) link=padre;//hijo de la raiz
23      else link=padre->get_link()->get_tran(pch);
24    }
25    return link;
26  }
27  trie* get_tran(int c) {
28    if(!tran[c])
29      tran[c] = !padre? this : this->get_link()->get_tran(c);
30    return tran[c];
31  }
32  trie *get_nxthoja(){
```

```
33     if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
34     return nxthoja;
35   }
36   void print(int p){
37     if(idhoja)
38       cout << "found " << idhoja << "  at position " << p-szhoja << endl
              ;
39     if(get_nxthoja()) get_nxthoja()->print(p);
40   }
41   void matching(const string &s, int p=0){
42     print(p);
43     if(p<sz(s)) get_tran(s[p])->matching(s, p+1);
```

5.   Geometria

5.1.   Punto

```
1 struct pto{
```

5.2.   Line

```
1 int sgn(ll x){return x<0? -1 : !!x;}
2 struct line{
3   line() {}
4   double a,b,c;//Ax+By=C
5 //pto MUST store float coordinates!
6   line(double a, double b, double c):a(a),b(b),c(c){}
7   line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
8   int side(pto p){return sgn(ll(a) * p.x + ll(b) * p.y - c);}
9 };
10 bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
11 pto inter(line l1, line l2){//intersection
12   double det=l1.a*l2.b-l2.a*l1.b;
13   if(abs(det)<EPS) return pto(INF, INF);//parallels
14   return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
15 }
```

5.3.   Segment

```
1 struct segm{
2   pto s,f;
3   segm(pto s, pto f):s(s), f(f) {}
4   pto closest(pto p) {//use for dist to point
5     double l2 = dist_sq(s, f);
```

```
6        if(l2==0.) return s;
7        double t =((p-s)*(f-s))/l2/l2;
8        if (t<0.) return s;//not write if is a line
9        else if(t>1.)return f;//not write if is a line
10       return s+((f-s)*t);
11     }
12     bool inside(pto p){return abs(dist(s, p)+dist(p, f)-dist(s, f))<EPS
           ;}
13 };
14
15 pto inter(segm s1, segm s2){
16   pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
17     if(s1.inside(r) && s2.inside(r))
18          return r;
19   return pto(INF, INF);
20 }
```

5.4.   Rectangle

```
1 struct rect{
2   //lower-left and upper-right corners
3   pto lw, up;
4 };
5 //returns if there's an intersection and stores it in r
6 bool inter(rect a, rect b, rect &r){
7   r.lw=pto(max(a.lw.x, b.lw.x), max(a.lw.y, b.lw.y));
8   r.up=pto(min(a.up.x, b.up.x), min(a.up.y, b.up.y));
9 //check case when only a edge is common
10  return r.lw.x<r.up.x && r.lw.y<r.up.y;
11 }
```

5.5.   Polygon Area

```
1 double area(vector<pto> &p){//O(sz(p))
2   double area=0;
3   forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
4   //if points are in clockwise order then area is negative
5   return abs(area)/2;
6 }
7 //Area ellipse = M_PI*a*b where a and b are the semi axis lengths
8 //Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2
```

5.6.   Circle

```
1 vec perp(vec v){return vec(-v.y, v.x);}
2 line bisector(pto x, pto y){
3   line l=line(x, y); pto m=(x+y)/2;
4   return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
5 }
6 struct Circle{
7   pto o;
8   double r;
9   Circle(pto x, pto y, pto z){
10    o=inter(bisector(x, y), bisector(y, z));
11    r=dist(o, x);
12  }
13  pair<pto, pto> ptosTang(pto p){
14    pto m=(p+o)/2;
15    tipo d=dist(o, m);
16    tipo a=r*r/(2*d);
17    tipo h=sqrt(r*r-a*a);
18    pto m2=o+(m-o)*a/d;
19    vec per=perp(m-o)/d;
20    return make_pair(m2-per*h, m2+per*h);
21  }
22 };
23 //finds the center of the circle containing p1 and p2 with radius r
24 //as there may be two solutions swap p1, p2 to get the other
25 bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
26      double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
27      if(det<0) return false;
28      c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
29      return true;
30 }
31 #define sqr(a) ((a)*(a))
32 #define feq(a,b) (fabs((a)-(b))<EPS)
33 pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
34   tipo dx = sqrt(b*b-4.0*a*c);
35   return make_pair((-b + dx)/(2.0*a),(-b - dx)/(2.0*a));
36 }
37 pair<pto, pto> interCL(Circle c, line l){
38   bool sw=false;
39   if((sw=feq(0,l.b))){
40   swap(l.a, l.b);
41   swap(c.o.x, c.o.y);
42   }
43   pair<tipo, tipo> rc = ecCuad(
```

```
44   sqr(l.a)+sqr(l.b),
45   2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
46   sqr(l.b)*(sqr(c.o.x)+sqr(c.o.y)-sqr(c.r))+sqr(l.c)-2.0*l.c*l.b*c.o.y
47   );
48   pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc.first) / l.b),
49           pto(rc.second, (l.c - l.a * rc.second) / l.b) );
50   if(sw){
51   swap(p.first.x, p.first.y);
52   swap(p.second.x, p.second.y);
53   }
54   return p;
55 }
56 pair<pto, pto> interCC(Circle c1, Circle c2){
57   line l;
58   l.a = c1.o.x-c2.o.x;
59   l.b = c1.o.y-c2.o.y;
60   l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o.x)+sqr(c1.o.y)
61   -sqr(c2.o.y))/2.0;
62   return interCL(c1, l);
63 }
```

5.7.   Point in Poly

```
1  //checks if v is inside of P, using ray casting
2  //works with convex and concave.
3  //excludes boundaries, handle it separately using segment.inside()
4  bool inPolygon(pto v, vector<pto>& P) {
5    bool c = false;
6    forn(i, sz(P)){
7      int j=(i+1)%sz(P);
8      if((P[j].y>v.y) != (P[i].y > v.y) &&
9    (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
10       c = !c;
11   }
12   return c;
13 }
```

5.8.   Convex Check CHECK

```
1  bool isConvex(vector<int> &p){//O(N)
2    int N=sz(p);
3    if(N<3) return false;
4    bool isLeft=p[0].left(p[1], p[2]);
5    forr(i, 1, N)
```

```
6     if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
7       return false;
8   return true; }
```

5.9.   Convex Hull

```
1  //stores convex hull of P in S, CCW order
2  void CH(vector<pto>& P, vector<pto> &S){
3    S.clear();
4    sort(P.begin(), P.end());
5    forn(i, sz(P)){
6      while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
7      S.pb(P[i]);
8    }
9    S.pop_back();
10   int k=sz(S);
11   dforn(i, sz(P)){
12     while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back
           ();
13     S.pb(P[i]);
14   }
15   S.pop_back();
16 }
```

5.10.   Cut Polygon

```
1  //cuts polygon Q along the line ab
2  //stores the left side (swap a, b for the right one) in P
3  void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
4    P.clear();
5    forn(i, sz(Q)){
6      double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
7      if(left1>=0) P.pb(Q[i]);
8      if(left1*left2<0)
9        P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
10   }
11 }
```

5.11.   Bresenham

```
1  //plot a line approximation in a 2d map
2  void bresenham(pto a, pto b){
3    pto d=b-a; d.x=abs(d.x), d.y=abs(d.y);
4    pto s(a.x<b.x? 1: -1, a.y<b.y? 1: -1);
```

```
5     int err=d.x-d.y;
6     while(1){
7       m[a.x][a.y]=1;//plot
8       if(a==b) break;
9       int e2=2*err;
10      if(e2 > -d.y){
11        err-=d.y, a.x+=s.x;
12      if(e2 < d.x)
13        err+= d.x, a.y+= s.y;
14    }
15  }
```

5.12.   Rotate Matrix

```
1  //rotates matrix t 90 degrees clockwise
2  //using auxiliary matrix t2(faster)
3  void rotate(){
4    forn(x, n) forn(y, n)
5      t2[n-y-1][x]=t[x][y];
6    memcpy(t, t2, sizeof(t));
7  }
```

5.13.   Interseccion de Circulos en n3log(n)

```
1  struct event {
2      double x; int t;
3      event(double xx, int tt) : x(xx), t(tt) {}
4      bool operator <(const event &o) const { return x < o.x; }
5  };
6  typedef vector<Circle> VC;
7  typedef vector<event> VE;
8  int n;
9  double cuenta(VE &v, double A,double B) {
10      sort(v.begin(), v.end());
11      double res = 0.0, lx = ((v.empty())?0.0:v[0].x);
12      int contador = 0;
13      forn(i,sz(v)) {
14          // interseccion de todos (contador == n), union de todos (
                 contador > 0),
15          // conjunto de puntos cubierto por exacta k Circulos (contador
                 == k)
16          if (contador == n) res += v[i].x - lx;
17          contador += v[i].t;
18          lx = v[i].x;
```

```
19      }
20      return res;
21  }
22  // Primitiva de sqrt(r*r - x*x) como funcion double de una variable x.
23  inline double primitiva(double x,double r) {
24      if (x >= r) return r*r*M_PI/4.0;
25      if (x <= -r) return -r*r*M_PI/4.0;
26      double raiz = sqrt(r*r-x*x);
27      return 0.5 * (x * raiz + r*r*atan(x/raiz));
28  }
29  double interCircle(VC &v) {
30      vector<double> p; p.reserve(v.size() * (v.size() + 2));
31      forn(i,sz(v)) {
32          p.push_back(v[i].c.x + v[i].r);
33          p.push_back(v[i].c.x - v[i].r);
34      }
35      forn(i,sz(v)) forn(j,i) {
36          Circle &a = v[i], b = v[j];
37          double d = (a.c - b.c).norm();
38          if (fabs(a.r - b.r) < d && d < a.r + b.r) {
39              double alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d
                     * a.r));
40              pto vec = (b.c - a.c) * (a.r / d);
41              p.pb((a.c + rotate(vec, alfa)).x);
42              p.pb((a.c + rotate(vec, -alfa)).x);
43          }
44      }
45      sort(p.begin(), p.end());
46      double res = 0.0;
47      forn(i,sz(p)-1) {
48          const double A = p[i], B = p[i+1];
49          VE ve; ve.reserve(2 * v.size());
50          forn(j,sz(v)) {
51              const Circle &c = v[j];
52              double arco = primitiva(B-c.c.x,c.r) - primitiva(A-c.c.x,c.r
                     );
53              double base = c.c.y * (B-A);
54              ve.push_back(event(base + arco,-1));
55              ve.push_back(event(base - arco, 1));
56          }
57          res += cuenta(ve,A,B);
58      }
59      return res;
```

```
60  }
```

6.   Math

6.1.   Identidades

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$
$$\sum_{i=0}^{n} i \binom{n}{i} = n * 2^{n-1}$$
$$\sum_{i=m}^{n} i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$$
$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$
$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$
$$\sum_{i=0}^{n} i(i-1) = \frac{8}{6}(\frac{n}{2})(\frac{n}{2}+1)(n+1) \text{ (doubles)} \rightarrow \text{Sino ver caso impar y par}$$
$$\sum_{i=0}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = [\sum_{i=1}^{n} i]^2$$
$$\sum_{i=0}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$
$$\sum_{i=0}^{n} i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^{p} \frac{B_k}{p-k+1} \binom{p}{k} (n+1)^{p-k+1}$$
$$r = e - v + k + 1$$

Teorema de Pick: (Area, puntos interiores y puntos en el borde)
$$A = I + \frac{B}{2} - 1$$

6.2.   Ec. Caracteristica

$$a_0 T(n) + a_1 T(n-1) + ... + a_k T(n-k) = 0$$
$$p(x) = a_0 x^k + a_1 x^{k-1} + ... + a_k$$
Sean $r_1, r_2, ..., r_q$ las raíces distintas, de mult. $m_1, m_2, ..., m_q$
$$T(n) = \sum_{i=1}^{q} \sum_{j=0}^{m_i-1} c_{ij} n^j r_i^n$$
Las constantes c_{ij} se determinan por los casos base.

6.3.   Combinatorio

```
1  forn(i, MAXN+1){//comb[i][k]=i tomados de a k
2    comb[i][0]=comb[i][i]=1;
3    forr(k, 1, i) comb[i][k]=(comb[i-1][k]+comb[i-1][k-1])%MOD;
4  }
5
6  ll lucas (ll n, ll k, int p){ //Calcula (n,k)%p teniendo comb[p][p]
       precalculado.
7    ll aux = 1;
8    while (n + k){
9      aux = (aux * comb[n%p][k%p]) %p;
10     n/=p, k/=p;
11   }
```

```
12   return aux;
13  }
```

6.4.   Exp. de Numeros Mod.

```
1  ll expmod (ll b, ll e, ll m){//O(log b)
2    if(!e) return 1;
3    ll q= expmod(b,e/2,m); q=(q*q)%m;
4    return e%2? (b * q)%m : q;
5  }
```

6.5.   Exp. de Matrices y Fibonacci en log(n)

```
1  #define SIZE 350
2  int NN;
3  void mul(double a[SIZE][SIZE], double b[SIZE][SIZE])
4  {
5      double res[SIZE][SIZE] = {{0}};
6      forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
7      forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
8  }
9  void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE])
10 {
11     forn(i, NN) forn(j, NN) res[i][j]=(i==j);
12     while(n){
13         if(n&1) mul(res, a), n--;
14         else mul(a, a), n/=2;
15     }
16 }
17
18 struct M22{       // |a b|
19   tipo a,b,c,d;// |c d|
20   M22 operator*(const M22 &p) const {
21     return (M22){a*p.a+b*p.c, a*p.b+b*p.d, c*p.a+d*p.c,c*p.b+d*p.d};}
22 };
23 M22 operator^(const M22 &p, int n){
24   if(!n) return (M22){1, 0, 0, 1};//identidad
25   M22 q=p^(n/2); q=q*q;
26   return n%2? p * q : q;}
27
28 ll fibo(ll n){//calcula el fibonacci enesimo
29   M22 mat=(M22){0, 1, 1, 1}^n;
30   return mat.a*f0+mat.b*f1;//f0 y f1 son los valores iniciales
31 }
```

6.6.   Matrices y determinante $O(n^3)$

```cpp
struct Mat {
    vector<vector<double> > vec;
    Mat(int n): vec(n, vector<double>(n) ) {}
    Mat(int n, int m): vec(n, vector<double>(m) ) {}
    vector<double> &operator[](int f){return vec[f];}
    const vector<double> &operator[](int f) const {return vec[f];}
    int size() const {return sz(vec);}
    Mat operator+(Mat &b) { ///this de n x m entonces b de n x m
        Mat m(sz(b),sz(b[0]));
        forn(i,sz(vec)) forn(j,sz(vec[0])) m[i][j] = vec[i][j] + b[i][j
            ];
        return m;
    }
    Mat operator*(const Mat &b) { ///this de n x m entonces b de m x t
        int n = sz(vec), m = sz(vec[0]), t = sz(b[0]);
        Mat mat(n,t);
        forn(i,n) forn(j,t) {
            forn(k,m)
                mat[i][j] += vec[i][k] * b[k][j];
        }
        return mat;
    }
    double determinant(){//sacado de e maxx ru
        double det = 1;
        int n = sz(vec);
        Mat m(*this);
        forn(i, n){//para cada columna
            int k = i;
            forr(j, i+1, n)//busco la fila con mayor val abs
                if(abs(m[j][i])>abs(m[k][i]))
                    k = j;
            if(abs(m[k][i])<1e-9) return 0;
            m[i].swap(m[k]);//la swapeo
            if(i!=k) det = -det;
            det *= m[i][i];
            forr(j, i+1, n) m[i][j] /= m[i][i];
            //hago 0 todas las otras filas
            forn(j, n) if (j!= i && abs(m[j][i])>1e-9)
                forr(k, i+1, n) m[j][k]-=m[i][k]*m[j][i];
        }
        return det;
    }
    Mat identidad(int n) {
        Mat m(n);
        forn(i,n) m[i][i] = 1;
        return m;
    }
    Mat transpuesta() {
        Mat m(sz(vec[0]),sz(vec));
        forn(i,sz(vec[0])) forn(j,sz(vec))
            m[i][j] = vec[j][i];
        return m;
    }
    void print() {
        int n = sz(vec), m = sz(vec[0]);
        cout << "**************" << endl;
        forn(i,n){
            forn(j,m) cout << "␣"+!j << vec[i][j];
            cout << endl;
        }
    }
};
```

6.7.   Teorema Chino del Resto

$$y = \sum_{j=1}^{n}(x_j * (\prod_{i=1,i\neq j}^{n} m_i)_{m_j}^{-1} * \prod_{i=1,i\neq j}^{n} m_i)$$

6.8.   Funciones de primos

```cpp
ll numPrimeFactors (ll n){
  ll rta = 0;
  map<ll,ll> f=fact(n);
  forall(it, f) rta += it->second;
  return rta;
}

ll numDiffPrimeFactors (ll n){
  ll rta = 0;
  map<ll,ll> f=fact(n);
  forall(it, f) rta += 1;
  return rta;
```

```
13  }
14
15  ll sumPrimeFactors (ll n){
16    ll rta = 0;
17    map<ll,ll> f=fact(n);
18    forall(it, f) rta += it->first;
19    return rta;
20  }
21
22  ll numDiv (ll n){
23    ll rta = 1;
24    map<ll,ll> f=fact(n);
25    forall(it, f) rta *= (it->second + 1);
26    return rta;
27  }
28
29  ll sumDiv (ll n){
30    ll rta = 1;
31    map<ll,ll> f=fact(n);
32    forall(it, f) rta *= ((ll)pow((double)it->first, it->second + 1.0)-1)
          / (it->first-1);
33    return rta;
34  }
35
36  ll eulerPhi (ll n){ // con criba: O(lg n)
37    ll rta = n;
38    map<ll,ll> f=fact(n);
39    forall(it, f) rta -= rta / it->first;
40    return rta;
41  }
42
43  ll eulerPhi2 (ll n){ // O (sqrt n)
44    ll r = n;
45    forr (i,2,n+1){
46      if ((ll)i*i > n)
47        break;
48      if (n % i == 0){
49        while (n%i == 0) n/=i;
50        r -= r/i;
51      }}
52    if (n != 1)
53      r-= r/n;
54    return r;
```

```
55  }
```

6.9.   Phollard's Rho (rolando)

```
1   ll gcd(ll a, ll b){return a?gcd(b %a, a):b;}
2
3   ll mulmod (ll a, ll b, ll c) { //returns (a*b)%c, and minimize overfloor
4     ll x = 0, y = a%c;
5     while (b > 0){
6       if (b % 2 == 1) x = (x+y) % c;
7       y = (y*2) % c;
8       b /= 2;
9     }
10    return x % c;
11  }
12
13  ll expmod (ll b, ll e, ll m){//O(log b)
14    if(!e) return 1;
15    ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
16    return e%2? mulmod(b,q,m) : q;
17  }
18
19  bool es_primo_prob (ll n, int a)
20  {
21    if (n == a) return true;
22    ll s = 0,d = n-1;
23    while (d % 2 == 0) s++,d/=2;
24
25    ll x = expmod(a,d,n);
26    if ((x == 1) || (x+1 == n)) return true;
27
28    forn (i, s-1){
29      x = mulmod(x, x, n);
30      if (x == 1) return false;
31      if (x+1 == n) return true;
32    }
33    return false;
34  }
35
36  bool rabin (ll n){ //devuelve true si n es primo
37    if (n == 1) return false;
38    const int ar[] = {2,3,5,7,11,13,17,19,23};
39    forn (j,9)
```

```
40    if (!es_primo_prob(n,ar[j]))
41        return false;
42   return true;
43 }
44
45 ll rho(ll n){
46     if( (n & 1) == 0 ) return 2;
47     ll x = 2 , y = 2 , d = 1;
48     ll c = rand() % n + 1;
49     while( d == 1 ){
50         x = (mulmod( x , x , n ) + c)%n;
51         y = (mulmod( y , y , n ) + c)%n;
52         y = (mulmod( y , y , n ) + c)%n;
53         if( x - y >= 0 ) d = gcd( x - y , n );
54         else d = gcd( y - x , n );
55     }
56     return d;
57 }
```

6.10. Criba

```
1  #define MAXP 100000 //no necesariamente primo
2  int criba[MAXP+1];
3  void crearcriba(){
4    int w[] = {4,2,4,2,4,6,2,6};
5    for(int p=25;p<=MAXP;p+=10) criba[p]=5;
6    for(int p=9;p<=MAXP;p+=6) criba[p]=3;
7    for(int p=4;p<=MAXP;p+=2) criba[p]=2;
8    for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
9      for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
10 }
11
12 vector<int> primos;
13 void buscarprimos(){
14   crearcriba();
15   forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
16 }
17
18 //~ Useful for bit trick:
19 //~ #define SET(i) ( criba[(i)>>5]|=1<<((i)&31) )
20 //~ #define INDEX(i) ( (criba[i>>5]>>((i)&31))&1 )
21 //~ unsigned int criba[MAXP/32+1];
```

6.11. Factorizacion

Sea $n = \prod p_i^{k_i}$, fact(n) genera un map donde a cada p_i le asocia su k_i

```
1  //factoriza bien numeros hasta MAXP^2
2  map<ll,ll> fact(ll n){ //O (cant primos)
3    map<ll,ll> ret;
4    forall(p, primos){
5      while(!(n%*p)){
6        ret[*p]++;//divisor found
7        n/=*p;
8      }
9    }
10   if(n>1) ret[n]++;
11   return ret;
12 }
13
14 //factoriza bien numeros hasta MAXP
15 map<ll,ll> fact2(ll n){ //O (lg n)
16   map<ll,ll> ret;
17   while (criba[n]){
18     ret[criba[n]]++;
19     n/=criba[n];
20   }
21   if(n>1) ret[n]++;
22   return ret;
23 }
24
25 map<ll,ll> f3;
26 void fact3(ll n){ //O (lg n)^3. un solo numero
27     if (n == 1) return;
28     if (rabin(n))
29         f3[n]++;
30     else{
31         ll aux = rho(n);
32         fact3(aux); fact3(n/aux);
33     }
34   if(n>1) f3[n]++;
35   return;
36 }
37
38 //Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
39 void divisores(const map<ll,ll> &f, vector<ll> &divs, map<ll,ll>::
       iterator it, ll n=1){
```

```
40    if(it==f.begin()) divs.clear();
41    if(it==f.end()) {
42        if(n>1) divs.pb(n);
43        return;
44    }
45    ll p=it->fst, k=it->snd; ++it;
46    forn(_, k+1)
47        divisores(f, divs, it, n), n*=p;
48 }
```

6.12. GCD

```
1 tipo gcd(tipo a, tipo b){return a?gcd(b %a, a):b;}
```

6.13. Extended Euclid

```
1 void extendedEuclid (ll a, ll b){ //a * x + b * y = d
2   if (!b) { x = 1; y = 0; d = a; return;}
3   extendedEuclid (b, a%b);
4   ll x1 = y;
5   ll y1 = x - (a/b) * y;
6   x = x1; y = y1;
7 }
```

6.14. LCM

```
1 tipo lcm(tipo a, tipo b){return a / gcd(a,b) * b;}
```

6.15. Inversos

```
1 #define MAXMOD 15485867
2 ll inv[MAXMOD];//inv[i]*i=1 mod MOD
3 void calc(int p){//O(p)
4   inv[1]=1;
5   forr(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
6 }
7 int inverso(int x){//O(log x)
8   return expmod(x, eulerphi(MOD)-2);//si mod no es primo(sacar a mano)
9   return expmod(x, MOD-2);//si mod es primo
10 }
```

6.16. Simpson

```
1 double integral(double a, double b, int n=10000) {//O(n), n=cantdiv
2   double area=0, h=(b-a)/n, fa=f(a), fb;
3   forn(i, n){
4     fb=f(a+h*(i+1));
5     area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
6   }
7   return area*h/6.;}
```

6.17. Fraction

```
1 tipo mcd(tipo a, tipo b){return a?mcd(b%a, a):b;}
2 struct frac{
3   tipo p,q;
4   frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
5   void norm(){
6     tipo a = mcd(p,q);
7     if(a) p/=a, q/=a;
8     else q=1;
9     if (q<0) q=-q, p=-p;}
10  frac operator+(const frac& o){
11    tipo a = mcd(q,o.q);
12    return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
13  frac operator-(const frac& o){
14    tipo a = mcd(q,o.q);
15    return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
16  frac operator*(frac o){
17    tipo a = mcd(q,o.p), b = mcd(o.q,p);
18    return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
19  frac operator/(frac o){
20    tipo a = mcd(q,o.q), b = mcd(o.p,p);
21    return frac((p/b)*(o.q/a),(q/a)*(o.p/b));}
22  bool operator<(const frac &o) const{return p*o.q < o.p*q;}
23  bool operator==(frac o){return p==o.p&&q==o.q;}
24 };
```

6.18. Polinomio

```
1 struct poly {
2     vector<tipo> c;//guarda los coeficientes del polinomio
3     poly(const vector<tipo> &c): c(c) {}
4     poly() {}
5   int gr(){//calculates grade of the polynomial
6     return sz(c); }
7   bool isnull() {return c.empty();}
```

```
8      poly operator+(const poly &o) const {
9          int m = sz(c), n = sz(o.c);
10         vector<tipo> res(max(m,n));
11         forn(i, m) res[i] += c[i];
12         forn(i, n) res[i] += o.c[i];
13         return poly(res);
14     }
15     poly operator*(const poly &o) const {
16         int m = sz(c), n = sz(o.c);
17         vector<tipo> res(m+n-1);
18         forn(i, m) forn(j, n) res[i+j]+=c[i]*o.c[j];
19         return poly(res);
20     }
21   tipo eval(tipo v) {
22       tipo sum = 0;
23       dforn(i, sz(c)) sum=sum*v + c[i];
24       return sum;
25   }
26     //poly contains only a vector<int> c (the coeficients)
27   //the following function generates the roots of the polynomial
28 //it can be easily modified to return float roots
29   set<tipo> roots(){
30       set<tipo> roots;
31       tipo a0 = abs(c[0]), an = abs(c[sz(c)-1]);
32       vector<tipo> ps,qs;
33       forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);
34       forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
35       forall(pt,ps)
36         forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
37             tipo root = abs((*pt) / (*qt));
38             if (eval(root)==0) roots.insert(root);
39         }
40       return roots;
41   }
42 };
43 poly interpolate(const vector<tipo> &x, const vector<tipo> &y) {
44     int n = sz(x);
45     poly p;
46     vector<tipo> aux(2);
47     forn(i, n) {
48         double a = y[i] - p.eval(x[i]);
49         forn(j, i) a /= x[i] - x[j];
50         poly add(vector<tipo>(1, a));
51         forn(j, i) aux[0]=-x[j], aux[1]=1, add = add*aux;
52         p = p + add;
53     }
54     return p;
55 }
56 //the following functions allows parsing an expression like
57 //34+150+4*45
58 //into a polynomial(el numero en funcion de la base)
59 #define LAST(s) (sz(s)? s[sz(s)-1] : 0)
60 #define POP(s) s.erase(--s.end());
61 poly D(string &s) {
62   poly d;
63   for(int i=0; isdigit(LAST(s)); i++) d.c.push_back(LAST(s)-'0'), POP(s)
         ;
64   return d;}
65
66 poly T(string &s) {
67   poly t=D(s);
68   if (LAST(s)=='*'){POP(s); return T(s)*t;}
69   return t;
70 }
71 //main function, call this to parse
72 poly E(string &s) {
73   poly e=T(s);
74   if (LAST(s)=='+'){POP(s); return E(s)+e;}
75   return e;
76 }
```

6.19.   Ec. Lineales

```
1 bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
2   int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
3   vector<int> p; forn(i,m) p.push_back(i);
4   forn(i, rw) {
5     int uc=i, uf=i;
6     forr(f, i, n) forr(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;
          uc=c;}
7     if (feq(a[uf][uc], 0)) { rw = i; break; }
8     forn(j, n) swap(a[j][i], a[j][uc]);
9     swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
10    tipo inv = 1 / a[i][i]; //aca divide
11    forr(j, i+1, n) {
12      tipo v = a[j][i] * inv;
```

```
13        forr(k, i, m) a[j][k]-=v * a[i][k];
14        y[j] -= v*y[i];
15      }
16    } // rw = rango(a), aca la matriz esta triangulada
17    forr(i, rw, n) if (!feq(y[i],0)) return false; // checkeo de
         compatibilidad
18    x = vector<tipo>(m, 0);
19    dforn(i, rw){
20      tipo s = y[i];
21      forr(j, i+1, rw) s -= a[i][j]*x[p[j]];
22      x[p[i]] = s / a[i][i]; //aca divide
23    }
24    ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
25    forn(k, m-rw) {
26      ev[k][p[k+rw]] = 1;
27      dforn(i, rw){
28        tipo s = -a[i][k+rw];
29        forr(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
30        ev[k][p[i]] = s / a[i][i]; //aca divide
31      }
32    }
33    return true;
34  }
```

6.20. FFT

```
1  //~ typedef complex<double> base; //menos codigo, pero mas lento
2  //elegir si usar complejos de c (lento) o estos
3  struct base{
4      double r,i;
5      base(double r=0, double i=0):r(r), i(i){}
6      double real()const{return r;}
7      void operator/=(const int c){r/=c, i/=c;}
8  };
9  base operator*(const base &a, const base &b){
10     return base(a.r*b.r-a.i*b.i, a.r*b.i+a.i*b.r);}
11 base operator+(const base &a, const base &b){
12     return base(a.r+b.r, a.i+b.i);}
13 base operator-(const base &a, const base &b){
14     return base(a.r-b.r, a.i-b.i);}
15 vector<int> rev;
16 vector<base> wlen_pw;
17 inline static void fft(base a[], int n, bool invert) {
18   forn(i, n) if(i<rev[i]) swap(a[i], a[rev[i]]);
19   for (int len=2; len<=n; len<<=1) {
20     double ang = 2*M_PI/len * (invert?-1:+1);
21     int len2 = len>>1;
22     base wlen (cos(ang), sin(ang));
23     wlen_pw[0] = base (1, 0);
24       forr(i, 1, len2) wlen_pw[i] = wlen_pw[i-1] * wlen;
25     for (int i=0; i<n; i+=len) {
26       base t,
27         *pu = a+i,
28         *pv = a+i+len2,
29         *pu_end = a+i+len2,
30         *pw = &wlen_pw[0];
31       for (; pu!=pu_end; ++pu, ++pv, ++pw) {
32         t = *pv * *pw;
33         *pv = *pu - t;
34         *pu = *pu + t;
35       }
36     }
37   }
38   if (invert) forn(i, n) a[i]/= n;
39 }
40 inline static void calc_rev(int n){//precalculo: llamar antes de fft!!
41     wlen_pw.resize(n);
42     rev.resize(n);
43     int lg=31-__builtin_clz(n);
44     forn(i, n){
45     rev[i] = 0;
46         forn(k, lg) if(i&(1<<k))
47             rev[i]|=1<<(lg-1-k);
48     }
49 }
50 inline static void multiply(const vector<int> &a, const vector<int> &b,
       vector<int> &res) {
51   vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
52     int n=1;
53   while(n < max(sz(a), sz(b))) n <<= 1;
54   n <<= 1;
55     calc_rev(n);
56   fa.resize (n),  fb.resize (n);
57   fft (&fa[0], n, false),  fft (&fb[0], n, false);
58   forn(i, n) fa[i] = fa[i] * fb[i];
59   fft (&fa[0], n, true);
```

```
60    res.resize(n);
61      forn(i, n) res[i] = int (fa[i].real() + 0.5);
62 }
63 void toPoly(const string &s, vector<int> &P){//convierte un numero a
      polinomio
64    P.clear();
65    dforn(i, sz(s)) P.pb(s[i]-'0');
66 }
```

6.21.   Tablas y cotas (Primos, Divisores, Factoriales, etc)

Factoriales

0! = 1	11! = 39.916.800
1! = 1	12! = 479.001.600 ($\in$ int)
2! = 2	13! = 6.227.020.800
3! = 6	14! = 87.178.291.200
4! = 24	15! = 1.307.674.368.000
5! = 120	16! = 20.922.789.888.000
6! = 720	17! = 355.687.428.096.000
7! = 5.040	18! = 6.402.373.705.728.000
8! = 40.320	19! = 121.645.100.408.832.000
9! = 362.880	20! = 2.432.902.008.176.640.000 ($\in$ tint)
10! = 3.628.800	21! = 51.090.942.171.709.400.000

max signed tint = 9.223.372.036.854.775.807

max unsigned tint = 18.446.744.073.709.551.615

Primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109
113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227
229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347
349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461
463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599
601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727
733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859
863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009
1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103
1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229
1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327
1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471
1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579
1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697
1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823
1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951
1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081

Primos cercanos a 10^n

9941 9949 9967 9973 10007 10009 10037 10039 10061 10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049 100057 100069
999959 999961 999979 999983 1000003 1000033 1000037 1000039
9999943 9999971 9999973 9999991 10000019 10000079 10000103 10000121
99999941 99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021 1000000033

Cantidad de primos menores que 10^n

$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$

$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$

$\pi(10^{10}) = 455.052,511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

Divisores

Cantidad de divisores (σ_0) para *algunos* $n/\neg\exists n' < n, \sigma_0(n') \geqslant \sigma_0(n)$

$\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$

$\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$

$\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$

$\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$

$\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$

Suma de divisores (σ_1) para *algunos* $n/\neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$

$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$

$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$

$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$

$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$

$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$

$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$

$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$

$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$

$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$

$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$

$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

7.   Grafos

7.1.   Dijkstra

```
1 #define INF 1e9
2 int N;
3 #define MAX_V 250001
4 vector<ii> G[MAX_V];
5 //To add an edge use
```

```
6   #define add(a, b, w) G[a].pb(make_pair(w, b))
7
8   ll dijkstra(int s, int t){//O(|E| log |V|)
9     priority_queue<ii, vector<ii>, greater<ii> > Q;
10    vector<ll> dist(N, INF); vector<int> dad(N, -1);
11    Q.push(make_pair(0, s)); dist[s] = 0;
12    while(sz(Q)){
13      ii p = Q.top(); Q.pop();
14      if(p.snd == t) break;
15      forall(it, G[p.snd])
16        if(dist[p.snd]+it->first < dist[it->snd]){
17          dist[it->snd] = dist[p.snd] + it->fst;
18          dad[it->snd] = p.snd;
19          Q.push(make_pair(dist[it->snd], it->snd));
20        }
21    }
22    return dist[t];
23    if(dist[t]<INF)//path generator
24      for(int i=t; i!=-1; i=dad[i])
25        printf("%d%c", i, (i==s?'\n':'␣'));
26  }
```

7.2.   Bellman-Ford

```
1   vector<ii> G[MAX_N];//ady. list with pairs (weight, dst)
2   int dist[MAX_N];
3   void bford(int src){//O(VE)
4     dist[src]=0;
5     forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
6       dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
7   }
8
9   bool hasNegCycle(){
10    forn(j, N) if(dist[j]!=INF) forall(it, G[j])
11      if(dist[it->snd]>dist[j]+it->fst) return true;
12    //inside if: all points reachable from it->snd will have -INF distance
          (do bfs)
13    return false;
14  }
```

7.3.   Floyd-Warshall

```
1   //G[i][j] contains weight of edge (i, j) or INF
2   //G[i][i]=0
```

```
3   int G[MAX_N][MAX_N];
4   void floyd(){//O(N^3)
5   forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
6     G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
7   }
8   bool inNegCycle(int v){
9     return G[v][v]<0;}
10  //checks if there's a neg. cycle in path from a to b
11  bool hasNegCycle(int a, int b){
12    forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
13      return true;
14    return false;
15  }
```

7.4.   Kruskal

```
1   struct Ar{int a,b,w;};
2   bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
3   vector<Ar> E;
4   ll kruskal(){
5       ll cost=0;
6       sort(E.begin(), E.end());//ordenar aristas de menor a mayor
7       uf.init(n);
8       forall(it, E){
9           if(uf.comp(it->a)!=uf.comp(it->b)){//si no estan conectados
10              uf.unir(it->a, it->b);//conectar
11              cost+=it->w;
12          }
13      }
14      return cost;
15  }
```

7.5.   Prim

```
1   bool taken[MAXN];
2   priority_queue<ii, vector<ii>, greater<ii> > pq;//min heap
3   void process(int v){
4       taken[v]=true;
5       forall(e, G[v])
6           if(!taken[e->second]) pq.push(*e);
7   }
8
9   ll prim(){
10      zero(taken);
```

```
11      process(0);
12      ll cost=0;
13      while(sz(pq)){
14          ii e=pq.top(); pq.pop();
15          if(!taken[e.second]) cost+=e.first, process(e.second);
16      }
17      return cost;
18  }
```

7.6. 2-SAT + Tarjan SCC

```
1   //We have a vertex representing a var and other for his negation.
2   //Every edge stored in G represents an implication. To add an equation
        of the form a||b, use addor(a, b)
3   //MAX=max cant var, n=cant var
4   #define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
5   vector<int> G[MAX*2];
6   //idx[i]=index assigned in the dfs
7   //lw[i]=lowest index(closer from the root) reachable from i
8   int lw[MAX*2], idx[MAX*2], qidx;
9   stack<int> q;
10  int qcmp, cmp[MAX*2];
11  //verdad[cmp[i]]=valor de la variable i
12  bool verdad[MAX*2+1];
13
14  int neg(int x) { return x>=n? x-n : x+n;}
15  void tjn(int v){
16    lw[v]=idx[v]=++qidx;
17    q.push(v), cmp[v]=-2;
18    forall(it, G[v]){
19      if(!idx[*it] || cmp[*it]==-2){
20        if(!idx[*it]) tjn(*it);
21        lw[v]=min(lw[v], lw[*it]);
22      }
23    }
24    if(lw[v]==idx[v]){
25      qcmp++;
26      int x;
27      do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
28      verdad[qcmp]=(cmp[neg(v)]<0);
29    }
30  }
31  //remember to CLEAR G!!!
```

```
32  bool satisf(){//O(n)
33    memset(idx, 0, sizeof(idx)), qidx=0;
34    memset(cmp, -1, sizeof(cmp)), qcmp=0;
35    forn(i, n){
36      if(!idx[i]) tjn(i);
37      if(!idx[neg(i)]) tjn(neg(i));
38    }
39    forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
40    return true;
41  }
```

7.7. Articulation Points

```
1   int N;
2   vector<int> G[1000000];
3   //V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
4   int qV, V[1000000], L[1000000], P[1000000];
5   void dfs(int v, int f){
6     L[v]=V[v]=++qV;
7     forall(it, G[v])
8       if(!V[*it]){
9         dfs(*it, v);
10        L[v] = min(L[v], L[*it]);
11        P[v]+= L[*it]>=V[v];
12      }
13      else if(*it!=f)
14        L[v]=min(L[v], V[*it]);
15  }
16  int cantart(){ //O(n)
17    qV=0;
18    zero(V), zero(P);
19    dfs(1, 0); P[1]--;
20    int q=0;
21    forn(i, N) if(P[i]) q++;
22  return q;
23  }
```

7.8. Comp. Biconexas y Puentas

```
1   struct edge {
2     int u,v, comp;
3     bool bridge;
4   };
5   vector<edge> e;
```

```
6  void addEdge(int u, int v) {
7    G[u].pb(sz(e)), G[v].pb(sz(e));
8    e.pb((edge){u,v,-1,false});
9  }
10 //d[i]=id de la dfs
11 //b[i]=lowest id reachable from i
12 int d[MAXN], b[MAXN], t;
13 int nbc;//cant componentes
14 int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
15 void initDfs(int n) {
16   zero(G), zero(comp);
17   e.clear();
18   forn(i,n) d[i]=-1;
19   nbc = t = 0;
20 }
21 stack<int> st;
22 void dfs(int u, int pe) {//O(n + m)
23   b[u] = d[u] = t++;
24   comp[u] = (pe != -1);
25   forall(ne, G[u]) if (*ne != pe){
26     int v = e[*ne].u ^ e[*ne].v ^ u;
27     if (d[v] == -1) {
28       st.push(*ne);
29       dfs(v,*ne);
30       if (b[v] > d[u]){
31         e[*ne].bridge = true; // bridge
32       }
33       if (b[v] >= d[u]){ // art
34         int last;
35         do {
36           last = st.top(); st.pop();
37           e[last].comp = nbc;
38         } while (last != *ne);
39         nbc++;
40         comp[u]++;
41       }
42       b[u] = min(b[u], b[v]);
43     }
44     else if (d[v] < d[u]) { // back edge
45       st.push(*ne);
46       b[u] = min(b[u], d[v]);
47     }
48   }
```

```
49 }
```

7.9.   LCA + Climb

```
1  const int MAXN=100001;
2  const int LOGN=20;
3  //f[v][k] holds the 2^k father of v
4  //L[v] holds the level of v
5  int N, f[MAXN][LOGN], L[MAXN];
6  //call before build:
7  void dfs(int v, int fa=-1, int lvl=0){//generate required data
8    f[v][0]=fa, L[v]=lvl;
9    forall(it, G[v])if(*it!=fa)
10     dfs(*it, v, lvl+1);
11 }
12 void build(){//f[i][0] must be filled previously, O(nlgn)
13   forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
14
15 #define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
16
17 int climb(int a, int d){//O(lgn)
18   if(!d) return a;
19   dforn(i, lg(L[a])+1)
20     if(1<<i<=d)
21       a=f[a][i], d-=1<<i;
22     return a;
23 }
24 int lca(int a, int b){//O(lgn)
25   if(L[a]<L[b]) swap(a, b);
26   a=climb(a, L[a]-L[b]);
27   if(a==b) return a;
28   dforn(i, lg(L[a])+1)
29     if(f[a][i]!=f[b][i])
30       a=f[a][i], b=f[b][i];
31   return f[a][0];
32 }
33 int dist(int a, int b) {//returns distance between nodes
34   return L[a]+L[b]-2*L[lca(a, b)];}
```

7.10.   Heavy Light Decomposition

```
1  int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
2  int dad[MAXN];//dad[v]=padre del nodo v
3  void dfs1(int v, int p=-1){//pre-dfs
```

```
4      dad[v]=p;
5      treesz[v]=1;
6      forall(it, G[v]) if(*it!=p){
7        dfs1(*it, v);
8        treesz[v]+=treesz[*it];
9      }
10  }
11  int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
12  //Las cadenas aparecen continuas en el recorrido!
13  int cantcad;
14  int homecad[MAXN];//dada una cadena devuelve su nodo inicial
15  int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
16  void heavylight(int v, int cur=-1){
17    if(cur==-1) homecad[cur=cantcad++]=v;
18    pos[v]=q++;
19    cad[v]=cur;
20    int mx=-1;
21    forn(i, sz(G[v])) if(G[v][i]!=dad[v])
22      if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
23    if(mx!=-1) heavylight(G[v][mx], cur);
24    forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
25      heavylight(G[v][i], -1);
26  }
27  //ejemplo de obtener el maximo numero en el camino entre dos nodos
28  //RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
29  //esta funcion va trepando por las cadenas
30  int query(int an, int v){//O(logn)
31    //si estan en la misma cadena:
32    if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
33    return max(query(an, dad[homecad[cad[v]]]),
34          rmq.get(pos[homecad[cad[v]]], pos[v]+1));
35  }
```

7.11.   Centroid Decomposition

```
1  int n;
2  vector<int> G[MAXN];
3  bool taken[MAXN];//poner todos en FALSE al principio!!
4  int padre[MAXN];//padre de cada nodo en el centroid tree
5
6  int szt[MAXN];
7  void calcsz(int v, int p) {
8    szt[v] = 1;
```

```
9      forall(it,G[v]) if (*it!=p && !taken[*it])
10       calcsz(*it,v), szt[v]+=szt[*it];
11  }
12  void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) {//O(nlogn)
13    if(tam==-1) calcsz(v, -1), tam=szt[v];
14    forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
15      {szt[v]=0; centroid(*it, f, lvl, tam); return;}
16    taken[v]=true;
17    padre[v]=f;
18    forall(it, G[v]) if(!taken[*it])
19      centroid(*it, v, lvl+1, -1);
20  }
```

7.12.   Euler Cycle

```
1  int n,m,ars[MAXE], eq;
2  vector<int> G[MAXN];//fill G,n,m,ars,eq
3  list<int> path;
4  int used[MAXN];
5  bool usede[MAXE];
6  queue<list<int>::iterator> q;
7  int get(int v){
8    while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
9    return used[v];
10  }
11  void explore(int v, int r, list<int>::iterator it){
12    int ar=G[v][get(v)]; int u=v^ars[ar];
13    usede[ar]=true;
14    list<int>::iterator it2=path.insert(it, u);
15    if(u!=r) explore(u, r, it2);
16    if(get(v)<sz(G[v])) q.push(it);
17  }
18  void euler(){
19    zero(used), zero(usede);
20    path.clear();
21    q=queue<list<int>::iterator>();
22    path.push_back(0); q.push(path.begin());
23    while(sz(q)){
24      list<int>::iterator it=q.front(); q.pop();
25      if(used[*it]<sz(G[*it])) explore(*it, *it, it);
26    }
27    reverse(path.begin(), path.end());
28  }
```

```
29  void addEdge(int u, int v){
30    G[u].pb(eq), G[v].pb(eq);
31    ars[eq++]=u^v;
32  }
```

7.13.   Diametro árbol

```
1   vector<int> G[MAXN];
2   int n,m;
3
4   int p[MAXN],d[MAXN],d2[MAXN];
5   int bfs(int r, int *d) {
6     queue<int> q;
7     d[r]=0; q.push(r);
8     int v;
9     while(sz(q)) { v=q.front(); q.pop();
10      forall(it,G[v]) if (d[*it]==-1) {
11        d[*it]=d[v]+1, p[*it]=v;
12        q.push(*it);
13      }
14    }
15    return v;//ultimo nodo visitado
16  }
17
18  vector<int> diams;
19  vector<ii> centros;
20  void diametros(){
21    memset(d,-1,sizeof(d));
22    memset(d2,-1,sizeof(d2));
23    diams.clear(), centros.clear();
24    forn(i, n) if(d[i]==-1){
25      int v,c;
26      c=v=bfs(bfs(i, d2), d);
27      forn(_,d[v]/2) c=p[c];
28      diams.pb(d[v]);
29      if(d[v]&1) centros.pb(ii(c, p[c]));
30      else centros.pb(ii(c, c));
31    }
32  }
```

7.14.   Chu-liu

```
1   void visit(graph &h, int v, int s, int r,
2     vector<int> &no, vector< vector<int> > &comp,
```

```
3     vector<int> &prev, vector< vector<int> > &next, vector<weight> &mcost,
4     vector<int> &mark, weight &cost, bool &found) {
5     if (mark[v]) {
6       vector<int> temp = no;
7       found = true;
8       do {
9         cost += mcost[v];
10        v = prev[v];
11        if (v != s) {
12          while (comp[v].size() > 0) {
13            no[comp[v].back()] = s;
14            comp[s].push_back(comp[v].back());
15            comp[v].pop_back();
16          }
17        }
18      } while (v != s);
19      forall(j,comp[s]) if (*j != r) forall(e,h[*j])
20        if (no[e->src] != s) e->w -= mcost[ temp[*j] ];
21    }
22    mark[v] = true;
23    forall(i,next[v]) if (no[*i] != no[v] && prev[no[*i]] == v)
24      if (!mark[no[*i]] || *i == s)
25        visit(h, *i, s, r, no, comp, prev, next, mcost, mark, cost, found)
            ;
26  }
27  weight minimumSpanningArborescence(const graph &g, int r) {
28      const int n=sz(g);
29    graph h(n);
30    forn(u,n) forall(e,g[u]) h[e->dst].pb(*e);
31    vector<int> no(n);
32    vector<vector<int> > comp(n);
33    forn(u, n) comp[u].pb(no[u] = u);
34    for (weight cost = 0; ;) {
35      vector<int> prev(n, -1);
36      vector<weight> mcost(n, INF);
37      forn(j,n) if (j != r) forall(e,h[j])
38        if (no[e->src] != no[j])
39          if (e->w < mcost[ no[j] ])
40            mcost[ no[j] ] = e->w, prev[ no[j] ] = no[e->src];
41      vector< vector<int> > next(n);
42      forn(u,n) if (prev[u] >= 0)
43        next[ prev[u] ].push_back(u);
44      bool stop = true;
```

```
45      vector<int> mark(n);
46      forn(u,n) if (u != r && !mark[u] && !comp[u].empty()) {
47        bool found = false;
48        visit(h, u, u, r, no, comp, prev, next, mcost, mark, cost, found);
49        if (found) stop = false;
50      }
51      if (stop) {
52        forn(u,n) if (prev[u] >= 0) cost += mcost[u];
53        return cost;
54      }
55    }
56  }
```

7.15.   Hungarian

```
1   #define MAXN 256
2   #define INFTO 0x7f7f7f7f
3   int n;
4   int mt[MAXN][MAXN]; // Matriz de costos (X * Y)
5   int xy[MAXN], yx[MAXN]; // Matching resultante (X->Y, Y->X)
6   int lx[MAXN], ly[MAXN], slk[MAXN], slkx[MAXN], prv[MAXN];
7   char S[MAXN], T[MAXN];
8   void updtree(int x) {
9     forn(y, n) if (lx[x] + ly[y] - mt[x][y] < slk[y]) {
10      slk[y] = lx[x] + ly[y] - mt[x][y];
11      slkx[y] = x;
12  } }
13  int hungar(){//Matching maximo de mayor costo en grafos dirigidos (N^3)
14    forn(i, n) {
15      ly[i] = 0;
16      lx[i] = *max_element(mt[i], mt[i]+n); }
17    memset(xy, -1, sizeof(xy));
18    memset(yx, -1, sizeof(yx));
19    forn(m, n) {
20      memset(S, 0, sizeof(S));
21      memset(T, 0, sizeof(T));
22      memset(prv, -1, sizeof(prv));
23      memset(slk, 0x7f, sizeof(slk));
24      queue<int> q;
25  #define bpone(e, p) { q.push(e); prv[e] = p; S[e] = 1; updtree(e); }
26      forn(i, n) if (xy[i] == -1) { bpone(i, -2); break; }
27      int x=0, y=-1;
28      while (y==-1) {
29        while (!q.empty() && y==-1) {
30          x = q.front(); q.pop();
31          forn(j, n) if (mt[x][j] == lx[x] + ly[j] && !T[j]) {
32            if (yx[j] == -1) { y = j; break; }
33            T[j] = 1;
34            bpone(yx[j], x);
35          }
36        }
37        if (y!=-1) break;
38        int dlt = INFTO;
39        forn(j, n) if (!T[j]) dlt = min(dlt, slk[j]);
40        forn(k, n) {
41          if (S[k]) lx[k] -= dlt;
42          if (T[k]) ly[k] += dlt;
43          if (!T[k]) slk[k] -= dlt;
44        }
45        forn(j, n) if (!T[j] && !slk[j]) {
46          if (yx[j] == -1) {
47            x = slkx[j]; y = j; break;
48          } else {
49            T[j] = 1;
50            if (!S[yx[j]]) bpone(yx[j], slkx[j]);
51          }
52        }
53      }
54      if (y!=-1) {
55        for(int p = x; p != -2; p = prv[p]) {
56          yx[y] = p;
57          int ty = xy[p]; xy[p] = y; y = ty;
58        }
59      } else break;
60    }
61    int res = 0;
62    forn(i, n) res += mt[i][xy[i]];
63    return res;
64  }
```

8.   Network Flow

8.1.   Dinic

```
1
2   const int MAX = 300;
```

```
3    int nodes, src, dst;
4    int dist[MAX], q[MAX], work[MAX];
5
6    struct Edge {
7        int to, rev;
8        ll f, cap;
9        Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(
             cap) {}
10   };
11   vector<Edge> G[MAX];
12
13   // Adds bidirectional edge
14   void addEdge(int s, int t, ll cap){
15       G[s].pb(Edge(t, sz(G[t]), 0, cap));
16       G[t].pb(Edge(s, sz(G[s])-1, 0, 0));
17   }
18
19   bool dinic_bfs(){
20       fill(dist, dist+nodes, -1);
21       dist[src]=0;
22       int qt=0;
23       q[qt++]=src;
24       for(int qh=0; qh<qt; qh++){
25           int u =q[qh];
26           forall(e, G[u]){
27               int v=e->to;
28               if(dist[v]<0 && e->f < e->cap){
29                   dist[v]=dist[u]+1;
30                   q[qt++]=v;
31               }
32           }
33       }
34       return dist[dst]>=0;
35   }
36
37   ll dinic_dfs(int u, ll f){
38       if(u==dst) return f;
39       for(int &i=work[u]; i<sz(G[u]); i++){
40           Edge &e = G[u][i];
41           if(e.cap<=e.f) continue;
42           int v=e.to;
43           if(dist[v]==dist[u]+1){
44               ll df=dinic_dfs(v, min(f, e.cap-e.f));
```

```
45               if(df>0){
46                   e.f+=df;
47                   G[v][e.rev].f-= df;
48                   return df;
49               }
50           }
51       }
52       return 0;
53   }
54
55   ll maxFlow(int _src, int _dst){
56       src=_src;
57       dst=_dst;
58       ll result=0;
59       while(dinic_bfs()){
60           fill(work, work+nodes, 0);
61           while(ll delta=dinic_dfs(src,INF))
62               result+=delta;
63       }
64       // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1
            forman el min-cut
65       return result;
66   }
```

8.2.   Konig

```
1    // asume que el dinic YA ESTA tirado
2    // asume que nodes-1 y nodes-2 son la fuente y destino
3    int match[maxnodes]; // match[v]=u si u-v esta en el matching, -1 si v
        no esta matcheado
4    int s[maxnodes]; // numero de la bfs del koning
5    queue<int> kq;
6    // s[e]%2==1  o si e esta en V1 y s[e]==-1-> lo agarras
7    void koning() {//O(n)
8      forn(v,nodes-2) s[v] = match[v] = -1;
9      forn(v,nodes-2) forall(it,g[v]) if (it->to < nodes-2 && it->f>0)
10       { match[v]=it->to; match[it->to]=v;}
11     forn(v,nodes-2) if (match[v]==-1) {s[v]=0;kq.push(v);}
12     while(!kq.empty()) {
13       int e = kq.front(); kq.pop();
14       if (s[e]%2==1) {
15         s[match[e]] = s[e]+1;
16         kq.push(match[e]);
```

```
17        } else {
18
19          forall(it,g[e]) if (it->to < nodes-2 && s[it->to]==-1) {
20            s[it->to] = s[e]+1;
21            kq.push(it->to);
22          }
23        }
24      }
25  }
```

8.3.   Edmonds Karp's

```
1   #define MAX_V 1000
2   #define INF 1e9
3   //special nodes
4   #define SRC 0
5   #define SNK 1
6   map<int, int> G[MAX_V];//limpiar esto
7   //To add an edge use
8   #define add(a, b, w) G[a][b]=w
9   int f, p[MAX_V];
10  void augment(int v, int minE){
11    if(v==SRC) f=minE;
12    else if(p[v]!=-1){
13      augment(p[v], min(minE, G[p[v]][v]));
14      G[p[v]][v]-=f, G[v][p[v]]+=f;
15    }
16  }
17  ll maxflow(){//O(VE^2)
18    ll Mf=0;
19    do{
20      f=0;
21      char used[MAX_V]; queue<int> q; q.push(SRC);
22      zero(used), memset(p, -1, sizeof(p));
23      while(sz(q)){
24        int u=q.front(); q.pop();
25        if(u==SNK) break;
26        forall(it, G[u])
27          if(it->snd>0 && !used[it->fst])
28            used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
29      }
30      augment(SNK, INF);
31      Mf+=f;
```

```
32      }while(f);
33      return Mf;
34  }
```

8.4.   Push-Relabel O(N3)

```
1   #define MAX_V 1000
2   int N;//valid nodes are [0...N-1]
3   #define INF 1e9
4   //special nodes
5   #define SRC 0
6   #define SNK 1
7   map<int, int> G[MAX_V];
8   //To add an edge use
9   #define add(a, b, w) G[a][b]=w
10  ll excess[MAX_V];
11  int height[MAX_V], active[MAX_V], count[2*MAX_V+1];
12  queue<int> Q;
13  void enqueue(int v) {
14    if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v); }
15  void push(int a, int b) {
16    int amt = min(excess[a], ll(G[a][b]));
17    if(height[a] <= height[b] || amt == 0) return;
18    G[a][b]-=amt, G[b][a]+=amt;
19    excess[b] += amt, excess[a] -= amt;
20    enqueue(b);
21  }
22  void gap(int k) {
23    forn(v, N){
24      if (height[v] < k) continue;
25      count[height[v]]--;
26      height[v] = max(height[v], N+1);
27      count[height[v]]++;
28      enqueue(v);
29    }
30  }
31  void relabel(int v) {
32    count[height[v]]--;
33    height[v] = 2*N;
34    forall(it, G[v])
35      if(it->snd)
36        height[v] = min(height[v], height[it->fst] + 1);
37    count[height[v]]++;
```

```
38      enqueue(v);
39  }
40  ll maxflow() {//O(V^3)
41    zero(height), zero(active), zero(count), zero(excess);
42    count[0] = N-1;
43    count[N] = 1;
44    height[SRC] = N;
45    active[SRC] = active[SNK] = true;
46    forall(it, G[SRC]){
47      excess[SRC] += it->snd;
48      push(SRC, it->fst);
49    }
50    while(sz(Q)) {
51      int v = Q.front(); Q.pop();
52      active[v]=false;
53    forall(it, G[v]) push(v, it->fst);
54    if(excess[v] > 0)
55      count[height[v]] == 1? gap(height[v]):relabel(v);
56    }
57    ll mf=0;
58    forall(it, G[SRC]) mf+=G[it->fst][SRC];
59    return mf;
60  }
```

8.5.   Min-cost Max-flow

```
1   const int MAXN=10000;
2   const ll INF = 1e14;
3
4   struct edge {
5     int u, v;
6     ll cap, cost, flow;
7     ll rem() { return cap - flow; }
8   };
9   int nodes;//numero de nodos
10  vector<int> G[MAXN];
11  vector<edge> e;
12  void addEdge(int u, int v, ll cap, ll cost) {
13    G[u].pb(sz(e)); e.pb((edge){u,v,cap,cost,0});
14    G[v].pb(sz(e)); e.pb((edge){v,u,0,-cost,0});
15  }
16  ll pot[MAXN], dist[MAXN], pre[MAXN], cap[MAXN];
17  ll mxFlow, mnCost;
```

```
18  void flow(int s, int t) {
19    fill(pot, pot+nodes, 0);
20    mxFlow=mnCost=0;
21    while(1){
22      fill(dist, dist+nodes, INF); dist[s] = 0;
23      fill(pre, pre+nodes, -1); pre[s]=0;
24      fill(cap, cap+nodes, 0); cap[s] = INF;
25      priority_queue<pair<ll,int> > q; q.push(make_pair(0,s));
26      //~ Bellman Ford
27      //~ forn(i,nodes) {
28        //~ forn(j,sz(e)) if (e[j].rem()) {
29          //~ ll c = e[j].cost + pot[e[j].u] - pot[e[j].v];
30          //~ if (dist[e[j].v] > dist[e[j].u] + c) {
31            //~ dist[e[j].v] = dist[e[j].u] + c;
32            //~ pre[e[j].v] = j;
33            //~ cap[e[j].v] = min(cap[e[j].u], e[j].rem());
34        //~ } }
35      //~ }
36      //~ Dijkstra
37      while (!q.empty()) {
38        pair<ll,int> top = q.top(); q.pop();
39        int u = top.second;
40        ll d = -top.first;
41        if (u == t) break;
42        if (d > dist[u]) continue;
43        forn(i,sz(G[u])) {
44          edge E = e[G[u][i]];
45          ll c = E.cost + pot[u] - pot[E.v];
46          if (E.rem() && dist[E.v] > dist[u] + c) {
47            dist[E.v] = dist[u] + c;
48            pre[E.v] = G[u][i];
49            cap[E.v] = min(cap[u], E.rem());
50            q.push(make_pair(-dist[E.v], E.v));
51          }
52        }
53      }
54      if (pre[t] == -1) break;
55      forn(u,nodes)
56        if (dist[u] == INF) pot[u] = INF;
57        else pot[u] += dist[u];
58      mxFlow +=cap[t];
59      mnCost +=cap[t]*pot[t];
60      for (int v = t; v != s; v = e[pre[v]].u) {
```

```
61        e[pre[v]].flow += cap[t];
62        e[pre[v]^1].flow -= cap[t];
63     }
64   }
65 }
```

9.   Template

```cpp
1  #include <bits/stdc++.h>
2  using namespace std;
3  #define dprint(v) cerr << #v"=" << v << endl //;)
4  #define forr(i,a,b) for(int i=(a); i<(b); i++)
5  #define forn(i,n) forr(i,0,n)
6  #define dforn(i,n) for(int i=n-1; i>=0; i--)
7  #define forall(it,v) for(typeof(v.begin()) it=v.begin();it!=v.end();++it
       )
8  #define sz(c) ((int)c.size())
9  #define zero(v) memset(v, 0, sizeof(v))
10 #define pb push_back
11 #define fst first
12 #define snd second
13 typedef long long ll;
14 typedef pair<int,int> ii;
15
16 int main() {
17   freopen("input.in", "r", stdin);
18     ios::sync_with_stdio(0);
19   while(){
20
21   }
22   return 0;
23 }
```

10.   Ayudamemoria

Cant. decimales

```cpp
1 #include <iomanip>
2 cout << setprecision(2) << fixed;
```

Rellenar con espacios(para justificar)

```cpp
1 #include <iomanip>
```

```cpp
2 cout << setfill('␣') << setw(3) << 2 << endl;
```

Leer hasta fin de linea

```cpp
1 #include <sstream>
2 //hacer cin.ignore() antes de getline()
3 while(getline(cin, line)){
4     istringstream is(line);
5     while(is >> X)
6       cout << X << "␣";
7     cout << endl;
8 }
```

Aleatorios

```cpp
1 #define RAND(a, b) (rand()%(b-a+1)+a)
2 srand(time(NULL));
```

Doubles Comp.

```cpp
1 const double EPS = 1e-9;
2 x == y   <=> fabs(x-y) < EPS
3 x >  y   <=> x > y + EPS
4 x >= y   <=> x > y - EPS
```

Limites

```cpp
1 #include <limits>
2 numeric_limits<T>
3   ::max()
4   ::min()
5   ::epsilon()
```

Muahaha

```cpp
1 #include <signal.h>
2 void divzero(int p){
3   while(true);}
4 void segm(int p){
5   exit(0);}
6 //in main
7 signal(SIGFPE, divzero);
8 signal(SIGSEGV, segm);
```

Mejorar velocidad

```
ios::sync_with_stdio(false);
```

Mejorar velocidad 2

```cpp
//Solo para enteros positivos
inline void Scanf(int& a){
  char c = 0;
  while(c<33) c = getc(stdin);
  a = 0;
  while(c>33) a = a*10 + c - '0', c = getc(stdin);
}
```

Expandir pila

```cpp
#include <sys/resource.h>
rlimit rl;
getrlimit(RLIMIT_STACK, &rl);
rl.rlim_cur=1024L*1024L*256L;//256mb
setrlimit(RLIMIT_STACK, &rl);
```

C++11

```
g++ --std=c++1
```

Leer del teclado

```cpp
freopen("/dev/tty", "a", stdin);
```

Iterar subconjunto

```cpp
for(int sbm=bm; sbm; sbm=(sbm-1)&bm)
```

File setup

```
//tambien se pueden usar comas: {a, x, m, l}
touch {a..l}.in; tee {a..l}.cpp < template.cpp
```